

# OAI EPC current development

Sebastien ROUX  
Eurecom  
sebastien.roux@eurecom.fr

July 16, 2013

# Contents

<b>MME and S+P-Gateway</b>	<b>3</b>
<b>S1AP layer</b>	<b>4</b>
<b>Inter-task interface</b>	<b>5</b>
3.1 Concurrency . . . . .	5
3.2 Overall description . . . . .	5
3.3 Signals from kernel . . . . .	6
3.4 Priority handling . . . . .	6
3.5 Message scheduling . . . . .	7
3.6 Message definition . . . . .	7
3.7 Task message handling . . . . .	8
3.8 Messages logging . . . . .	8
3.9 Limitations . . . . .	9
3.10 Benefits . . . . .	9
3.11 To do . . . . .	9
<b>Signal API</b>	<b>10</b>
<b>Timer API</b>	<b>11</b>
5.1 Timer types . . . . .	11
5.2 Requesting a new signal . . . . .	11
5.3 Disable and remove a timer . . . . .	11
5.4 Timer signal expiry . . . . .	12
<b>Compiling core EPC</b>	<b>13</b>
6.1 Dependencies . . . . .	13
6.1.1 ASN1c . . . . .	13
6.1.2 libnettle . . . . .	13
6.1.3 gnutls . . . . .	14
6.1.4 freediameter . . . . .	14
<b>System configuration</b>	<b>16</b>
7.1 Global MME parameters . . . . .	16
7.1.1 Relative MME capacity . . . . .	16
7.1.2 Maximum number of UE . . . . .	16
7.1.3 Maximum number of eNB . . . . .	16
7.1.4 Tracking Area Identity . . . . .	17
7.1.5 MME Code . . . . .	17

7.1.6	MME Group Id . . . . .	17
7.2	Intertask parameters . . . . .	17
7.2.1	Queue size per task . . . . .	17
7.3	SCTP parameters . . . . .	17
7.3.1	IN/OUT streams number . . . . .	17
7.4	S1AP parameters . . . . .	18
7.4.1	Outcome timer . . . . .	18
7.5	Network interfaces parameters . . . . .	18

# List of Abbreviations

API	Application Programming Interface
ASN.1	Abstract Syntax Notation.1
CPU	Control Processing Unit
IE	Informations Elements
MCC	Mobile Country Code
MMEC	MME Code
MMEGID	MME Group Id
MNC	Mobile Network Code
PDU	Protocol Data Unit
S1AP	S1 Application Protocol
TAC	Tracking Area Code
TAI	Tracking Area Identity
XML	Extensible Markup Language

# MME and S+P-Gateway

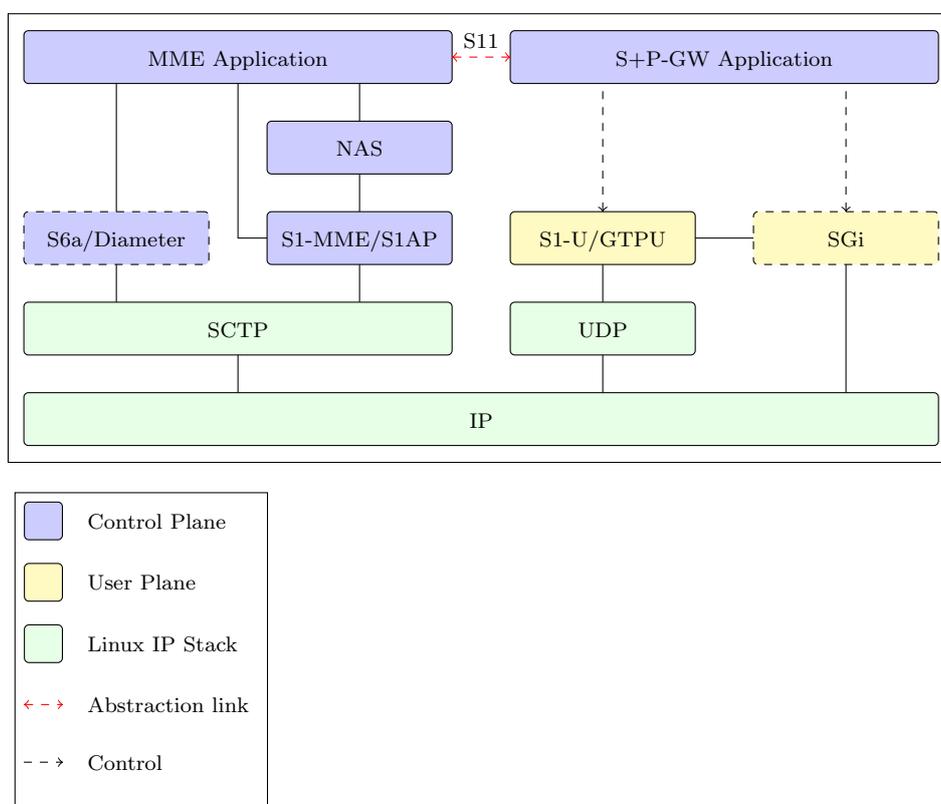


Figure A: MME and S+P-GW architecture

Figure A shows the targeted implementation of EPC.

# S1AP layer

S1AP layer relies on ASN.1 messages description. Generating C code from the specification implies three steps:

1. Modify .asn files to match tools limitation.
2. Generate IEs with the help of the asn1c free tool.
3. Use the provided script which generates PDU codec<sup>1</sup>.

---

<sup>1</sup>Encode/Decode

# Inter-task interface

## 3.1 Concurrency

Code divided in layers should be able to be executed in parallel to use the full CPUs power. We can achieve better performance by using some mechanisms that runs part of code in parallel on UNIX platforms:

- Multi-process: no link between processes, usage of sockets or pipes is mandatory.
- Forks: code is duplicated and data are stored in different spaces.
- Threads: only code is duplicated, data space is shared between processes

Data synchronization is the first issue to think about when using such mechanisms.

## 3.2 Overall description

A single API (called ITTI<sup>1</sup>) is used to manage messages exchanged between the tasks which are running on separate threads. This will lead in better usage of multi-core environments. Figure B describes the basic fonctionnement of the

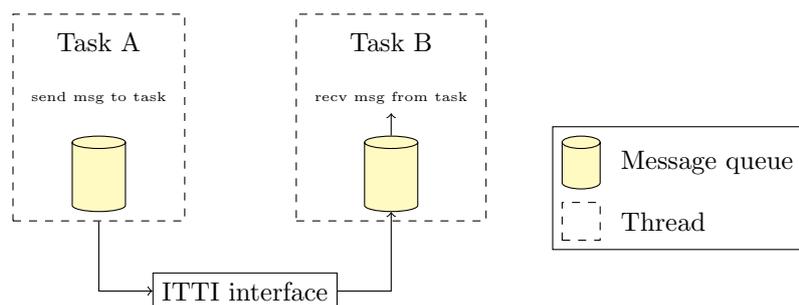


Figure B: Overall process

inter-task interface. A message sent from Task A is en-queued to the message queue belonging to the target task. Note that tasks can send messages to

<sup>1</sup>InTer-Task Interface

themselves. Another API (See Figure C) defines broadcast messaging where any task can send a broadcast message to every other task.

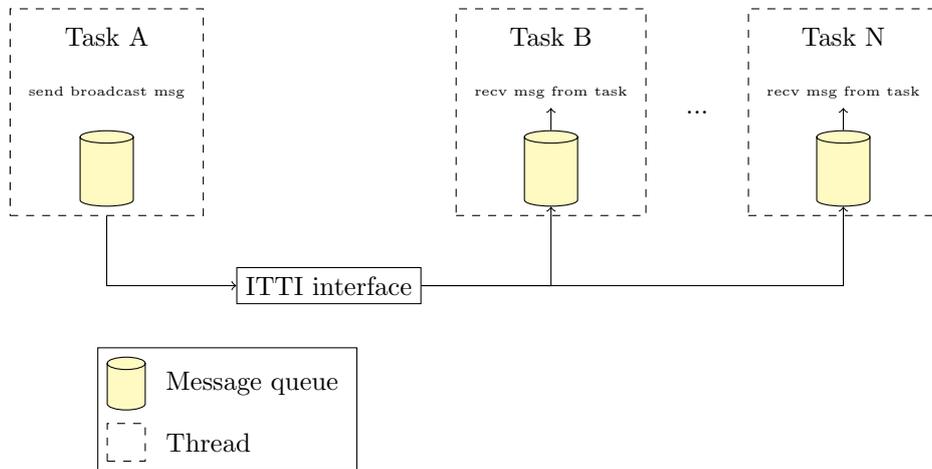


Figure C: Broadcasting process

Once a task received a new message and if the task is in sleep mode (i.e. not handling any message), the task is waken up and the message is de-queued. We can imagine a limit in number of parallel tasks, for example  $N+1$  CPU's. Architectures using hyper-threading mechanism can have this value extended to  $2 \times$  CPU's. Every task is running in a separated pthread, awaiting for new messages to handle.

### 3.3 Signals from kernel

Handling of signals from kernel is the more critical part as a signal can be raised at any moment and will interrupt one of the running thread. Used signals should be restricted to a single task that will handle them (using the POSIX sigmask function). Moreover, it isn't thread-safe to use mutexes inside a signal handler, a synchronization flag should be used to notify the signal handler task. This task will then send the appropriate message to the right task. For example, handling of signals can be done in the main thread or by a background task scheduled periodically. To overcome these issues, sigtimedwait and sigwaitinfo API functions can be used to wait for a signal to happen. Signals will be received in the thread context, as far as other threads block these signals.

### 3.4 Priority handling

Usage of message prioritization enables tasks to send critical messages with a faster delivery time regarding other messages en-queued for the target task. For now only seven priority levels can be applied when defining tasks:

- TASK\_PRIORITY\_MAX

- TASK\_PRIORITY\_MAX\_LEAST
- TASK\_PRIORITY\_MED\_PLUS
- TASK\_PRIORITY\_MED
- TASK\_PRIORITY\_MED\_LEAST
- TASK\_PRIORITY\_MIN\_PLUS
- TASK\_PRIORITY\_MIN

For now message priority does not involve any suitable scheduler: every time a message is de-queued, message priority of other messages is incremented by one.

### 3.5 Message scheduling

Currently, there is no software limit on the maximum number of threads executed in parallel. When a task sends a message, it is en-queued in the right message queue, belonging to the target task. The queue is a double-linked list. A mutex prevents other tasks from modifying this queue while a task is en-queueing or removing a message.

### 3.6 Message definition

Messages are defined using a single macro that adds the message to the ids enumeration and maps data of the message to the union of messages.

```
1 MESSAGE_DEF(S1AP_SCTP_NEW_MESSAGE_IND,
             TASK_PRIORITY_MED, S1apSctpNewMessageInd
             s1apSctpNewMessageInd)
```

and the associated data:

```
1 typedef struct {
2     uint8_t *buffer;          ///< SCTP buffer
3     uint32_t bufLen;         ///< SCTP buffer length
4     int32_t  assocId;        ///< SCTP physical
                             association ID
5     uint8_t  stream;         ///< Stream number on which
                             data had been received
6     uint16_t instreams;      ///< Number of input streams
                             for the SCTP connection between peers
7     uint16_t outstreams;     ///< Number of output
                             streams for the SCTP connection between peers
8 } S1apSctpNewMessageInd;
```

### 3.7 Task message handling

```
1 void* slap_mme_thread(void *args) {
2     while(1) {
3         /* Trying to fetch a message from the message
4            queue.
5            * If the queue is empty, this function will
6            block till a
7            * message is sent to the task.
8            */
9         receive_msg(TASK_S1AP, &receivedMessage);
10        assert(receivedMessage != NULL);
11        switch(receivedMessage->messageId) {
12            case S1AP_SCTP_NEW_MESSAGE_IND:
13            {
14                //Some processing
15            } break;
16            default:
17            {
18                S1AP_DEBUG("Unkwnon message ID %d\n",
19                    receivedMessage->messageId);
20            } break;
21        }
22        free(receivedMessage);
23        receivedMessage = NULL;
24    }
25 }
```

### 3.8 Messages logging

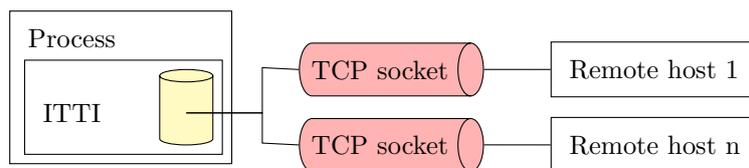


Figure D: Remote debugging

Logging of inter-tasks messages can be setup using an external tool that will be connected to the ITTI. Based on an array of dumped messages, they are serialized to produce an array of byte sent over a socket. A remote tool can then decode messages and display fields, message number, time. Additionally, logs from standard output can be printed over the debug tool. Multi-user debugging on only one running process can be achieved using this interface. Messages to dump should be queued for a pure asynchronous communication between the dump task and the remote hosts. Another interesting feature could be to send a message to a task from an host, allowing run-time

re-configuration. The C pre-processor can be used to generate messages definition (using XML templates for example).

### **3.9 Limitations**

Data pointers belonging to one task should never be passed to another task. The presented mechanism does not prevent a task from being locked. In such a case, the blocked task will no more handle messages incoming from other tasks.

### **3.10 Benefits**

- Only a single entry point between all tasks (easy inter-task communication tracking and debugging).
- Usage of message queues enables parallelization of layers.
- Message prioritization and scheduling.
- Protection of data between threads is done by the API at an higher level.

### **3.11 To do**

- Implement a priority based scheduler. Currently the queue of messages works as a FIFO.
- Limit the number of tasks thread that can be run in parallel

# Signal API

On LINUX platforms, processes will receive signals coming from Kernel. A single blocking entry point handles all used signals that are requested by the MME. The main application thread is reserved to signal handling as this thread will be blocked till a new signal is ready for handling.

Using this method prevents threads from being interrupted by signals handler which can interrupt the thread at any time and as a consequence create some misbehaving in threads contexts. Following is a sample list of signals handled:

- SIGABRT This is signal is sent to the process when `abort()` function is called within the process and kill the process. Process can for example display the stack once this signal is received.
- SIGRTMIN This signal is used by the timer API and is raised everytime a timer has expired.

Till now there is no way for tasks to request a new signal.

# Timer API

Timer API doesn't consist of a task (i.e. tasks cannot send messages to it). Handling of UNIX signal associated to the timers is a Real-time signal with an id (SIGRTMIN) depending on the platform. Management of this signal is done by the signal interface and developers should not care about handling timer signals incoming from Kernel. Once a timer has expired the task which has requested it will receive the `TIMER_HAS_EXPIRED` signal. Note that `timer_id` is of type `long` and thus its size is platform specific.

## 5.1 Timer types

- `TIMER_ONE_SHOT` After expiry and its associated signal, the timer is removed.
- `TIMER_PERIODIC` The timer is automatically reloaded on each expiry while the task which has setup this timer doesn't cancel it.

## 5.2 Requesting a new signal

Any task can request a new signal by invoking the following API:

```
1 int timer_setup(  
2     uint32_t      interval_sec ,  
3     uint32_t      interval_us ,  
4     task_id_t     task_id ,  
5     timer_type_t  type ,  
6     long          *timer_id);
```

Note that `timer_id` is a unique identifier to distinguish timers.

## 5.3 Disable and remove a timer

Disable and remove the timer referenced by `timer_id`.

```
1 int timer_remove(long timer_id);
```

## 5.4 Timer signal expiry

Once the signal dispatcher receives the SIGRTMIN signal, a new signal is sent to the task which has requested the timer. Contrary to signal request, timer expiry notification is achieved using the intertask mechanism. The signal data associated to this event follows:

```
1 typedef struct {  
2     long timer_id;  
3 } timer_has_expired_t;
```

# Compiling core EPC

The core EPC software has been tested on Ubuntu 12.04LTS x86 and ia64. Before compiling the core EPC, some packages should be installed on the platform.

## 6.1 Dependencies

- libstcp-dev
- libpthread-dev
- automake and autoconf
- libtoolize
- gcc, g++, make
- flex and bison
- openssl-dev
- asnlc (see section 6.1.1)
- libnettle (see section 6.1.2)
- freediameter (see section 6.1.4) and gnutls 3.1.0 (see section 6.1.3)

Command-line to install the required packages:

```
1 sudo apt-get install cmake make gcc flex bison \  
2 libstcp1 libstcp-dev libidn2-0-dev libidn11-dev \  
3 libmysqlclient-dev libxml2-dev swig python-dev \  
4 cmake-curses-gui valgrind guile-2.0-dev \  
5 libgmp-dev libgcrypt11-dev gdb unzip \  
6 libtasn1-3-dev g++ autoconf automake \  
7 openssl-dev -y
```

### 6.1.1 ASN1c

### 6.1.2 libnettle

The nettle library is used by freediameter for certificate encryption and by core EPC for key derivation.

```

1 wget ftp://ftp.lysator.liu.se/pub/security/lsh/nettle
   -2.5.tar.gz
2 gunzip nettle-2.5.tar.gz
3 tar -xvf nettle-2.5.tar
4 cd nettle-2.5/
5 ./configure --disable-openssl --enable-shared
6 make
7 make check
8 sudo make install

```

The commands provided above will download the required packages sources, configure and install them on the system. Note that any packages which is not in the Ubuntu repository is installed by default in `/usr/local` instead of `/usr`. This behaviour can be overridden at configuration time by providing `-prefix=/usr` to the configuration script (`configure`).

### 6.1.3 gnutls

The GNUTls library is only used by freediameter for certificate handling and as a consequence should be installed before trying to compile the freediameter library.

```

1 wget ftp://ftp.gnutls.org/gcrypt/gnutls/v3.1/gnutls
   -3.1.0.tar.xz
2 tar -xvf gnutls-3.1.0.tar.xz
3 cd gnutls-3.1.0/
4 ./configure LDFLAGS='-L/usr/local/lib'
5 make
6 sudo make install

```

Note: when dependencies are installed in `/usr/local` instead of `/usr`, `LDFLAGS` has to be overridden with the path to libraries when configuring the package: `LDFLAGS='-L/usr/local/lib'`.

### 6.1.4 freediameter

Freediameter is the package that provides diameter capabilities to the MME/HSS. On top of this stack, S6A avp dictionary is used to enable a compliant S6A interface.

```

1 wget http://www.freediameter.net/hg/freeDiameter/
   archive/1.1.5.tar.gz
2 tar -xvf 1.1.5.tar.gz
3 cd freeDiameter-1.1.5
4 patch -p1 < ../../freediameter-1.1.5.patch
5 mkdir build
6 cd build
7 cmake ../
8 make
9 make test
10 sudo make install

```

If you want to install this package in /usr instead of /usr/local,  
-DCMAKE\_INSTALL\_PREFIX:PATH=/usr should be passed to cmake at configuration.

# System configuration

Currently there is two ways to configure the system:

- Compilation configuration
- Boot-up configuration

In the first type of configuration, the single system configuration structure is filled in with default values that can be found in the `mme_default_values.h` header file.

When Boot-up configuration is used, a configuration file is passed to the process by using the either `-c filename.conf` or `-conf=filename.conf`. This file is then parsed by the bison interpreter and values are replaced in the global system configuration structure.

## 7.1 Global MME parameters

### 7.1.1 Relative MME capacity

Even though this parameter is not used by the MME for controlling the MME load balancing within a pool (at least for now), the parameter has to be forwarded to the eNBs during association procedure. This parameter is encoded on 8bits, acceptable values going from 0 to 255. (Default value = 15)

```
1 RELATIVE_CAPACITY = 10;
```

### 7.1.2 Maximum number of UE

This limit is present here only for debug purposes and is used to restrict the number of served UE the MME can handle. In real network another mechanism will trigger an MME overload for certain eNBs and will restrict certain types of traffic. Such a mechanism would imply the Relative MME capacity.

```
1 MAXUE = 100;
```

### 7.1.3 Maximum number of eNB

Refer to 7.1.2.

```
1 MAXENB = 10;
```

### 7.1.4 Tracking Area Identity

TAI is the concatenation of MCC, MNC and TAC. The TAC uniquely identifies a PLMN within a Cell Id.

```
1 PLMN = mcc.mnc:tac;
```

Multiple values can be given using a comma separator. Example:

```
1 PLMN = 208.38:0,209.130:4,208.35:8;
```

### 7.1.5 MME Code

A list of a maximum of 256 values can be provided. MME Code is encoded on 8 bits, so acceptable range is: 0 to 255. Example:

```
1 MME_CODE = 30,56,1,8;
```

### 7.1.6 MME Group Id

A list of a maximum of 65536 values can be provided. MME Group Id is encoded on 16 bits, so acceptable range is: 0 to 65535. Example:

```
1 MME_GID = 3,4,5,30,8,9,50021;
```

## 7.2 Intertask parameters

### 7.2.1 Queue size per task

To restrict the number of messages in queues or to detect a possible MME overload, an upper bound for the queue size can be defined like this:

```
1 ITTI_QUEUE_SIZE = 2000000;
```

This parameter is expressed in bytes. Note that all messages exchanged by tasks have the same size.

## 7.3 SCTP parameters

### 7.3.1 IN/OUT streams number

The number of input/output streams can be configured to limit the number of streams used for UE-associated signalling. Note that stream with id = 0 is reserved for non-UE associated signalling. At least two streams should be used by the MME. (Default value = 64/64 streams)

```
1 SCTP_INSTREAMS = 32;  
2 SCTP_OUTSTREAMS = 32;
```

## 7.4 S1AP parameters

### 7.4.1 Outcome timer

Once an outcome is sent from MME to eNB, the MME locally starts a timer to abort the procedure and release UE contexts if the expected answer to this outcome is not received at the expiry of this timer.

This timer is expressed in seconds. (Default value = 5 seconds)

```
1 S1AP_OUTCOME_TIMER = 10;
```

## 7.5 Network interfaces parameters

Three parameters can be tuned in the configuration file:

- Interface Name: The related interface will be bind to this interface name
- IP address: Currently only IPv4 address is allowed
- IP netmask: Netmask for the LAN

These three parameters can be setup for five different interfaces used:

- SGW interface for S11
- SGW interface for S1U/S12/S4 in user plane
- SGW interface for S5/S8 in user plane
- PGW interface for S5/S8
- PGW interface for SGi
- MME interface for S1-MME in control plane

Example of configuration:

```
1 # ----- Interfaces definitions
2 SGW_INTERFACE_NAME_FOR_S11           = "s11sgw";
3 SGW_IP_ADDRESS_FOR_S11               = "192.168.10.1"
4 ;
5 SGW_IP_NETMASK_FOR_S11                = 24;
6
7 SGW_INTERFACE_NAME_FOR_S1U_S12_S4_UP = "upsgw0";
8 SGW_IP_ADDRESS_FOR_S1U_S12_S4_UP    = "192.168.1.1";
9 SGW_IP_NETMASK_FOR_S1U_S12_S4_UP    = 24;
10
11 SGW_INTERFACE_NAME_FOR_S5_S8_UP      = "upsgw1";
12 SGW_IP_ADDRESS_FOR_S5_S8_UP         = "192.168.5.2";
13 SGW_IP_NETMASK_FOR_S5_S8_UP         = 24;
14
15 PGW_INTERFACE_NAME_FOR_S5_S8         = "uppgw0";
16 PGW_IP_ADDRESS_FOR_S5_S8            = "192.168.5.1";
17 PGW_IP_NETMASK_FOR_S5_S8            = 24;
```

```
17
18 PGW_INTERFACE_NAME_FOR_SGI           = "eth1";
19 PGW_IP_ADDR_FOR_SGI                   = "192.168.12.30
    ";
20 PGW_IP_NETMASK_FOR_SGI                 = 24;
21
22 MME_INTERFACE_NAME_FOR_S1_MME          = "cpmme0";
23 MME_IP_ADDRESS_FOR_S1_MME              = "192.168.11.1"
    ;
24 MME_IP_NETMASK_FOR_S1_MME              = 24;
```